Data Preprocessing

May 19, 2020

0.0.1 Import libraries for the funcationality requirements.

some examples such as numpy, matplotlib, pandas

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

0.0.2 Getting the dataset that is to be processed

The data is downloaded from the website https://www.superdatascience.com

[50]:
```
DATA_loc = "Data_pre_data.csv"
```

[51]:
```
dataset = pd.read_csv(DATA_loc)
```

[52]:
```
dataset
```

[52]:
```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
5   France  35.0  58000.0       Yes
6    Spain   NaN  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

0.0.3 spliting x and y as independent and dependent variable

[53]:
```
X = dataset.iloc[:,:-1].values
```

[54]:
```
X
```

[54]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
```

```
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

[55]:
```
y = dataset.iloc[:,3].values
```

[56]:
```
y
```

[56]:
```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

0.0.4 Handling Missing Data

[57]:
```
from sklearn.preprocessing import Imputer
```

[58]:
```
imputer = Imputer(missing_values = "NaN", strategy = 'mean', axis = 0)
```

[59]:
```
imputer = imputer.fit(X[:, 1:3])
```

[60]:
```
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

[61]:
```
X
```

[61]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

0.0.5 Encoding and understanding Categorical Data

[62]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
# class for labeling the data
```

[65]:
```
labelencoder_X = LabelEncoder()
```

[66]:
```
X[:, 0] = labelencoder_X.fit_transform(X[:,0])
```

[67]:
```
X
```

[67]:
```
array([[0, 44.0, 72000.0],
       [2, 27.0, 48000.0],
       [1, 30.0, 54000.0],
       [2, 38.0, 61000.0],
```

```
       [1, 40.0, 63777.77777777778],
       [0, 35.0, 58000.0],
       [2, 38.77777777777778, 52000.0],
       [0, 48.0, 79000.0],
       [1, 50.0, 83000.0],
       [0, 37.0, 67000.0]], dtype=object)
```

[68]:
```
onehotencoder = OneHotEncoder(categorical_features=[0])
```

[69]:
```
X = onehotencoder.fit_transform(X).toarray()
```

```
/home/akki/.conda/envs/ml/lib/python3.5/site-
packages/sklearn/preprocessing/_encoders.py:363: FutureWarning: The handling of
integer data will change in version 0.22. Currently, the categories are
determined based on the range [0, max(values)], while in the future they will be
determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the
categories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
```

[70]:
```
X
```

[70]:
```
array([[1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.40000000e+01,
        7.20000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 2.70000000e+01,
        4.80000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 3.00000000e+01,
        5.40000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.80000000e+01,
        6.10000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.00000000e+01,
        6.37777778e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.50000000e+01,
        5.80000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.87777778e+01,
        5.20000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.80000000e+01,
        7.90000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 5.00000000e+01,
        8.30000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.70000000e+01,
        6.70000000e+04]])
```

[71]:
```
labelencoder_y = LabelEncoder()
```

[72]:
```
y = labelencoder_y.fit_transform(y)
```

[73]:
```
y
```

[73]: array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])

0.0.6 Spliting data into test and train set

[74]:
```
from sklearn.model_selection import train_test_split
```

[85]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 1)
```

[86]:
```
X_train
```

[86]:
```
array([[0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.87777778e+01,
        5.20000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.00000000e+01,
        6.37777778e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.40000000e+01,
        7.20000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.80000000e+01,
        6.10000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 2.70000000e+01,
        4.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.80000000e+01,
        7.90000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 5.00000000e+01,
        8.30000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.50000000e+01,
        5.80000000e+04]])
```

[87]:
```
y_train
```

[87]: array([0, 1, 0, 0, 1, 1, 0, 1])

[88]:
```
X_test
```

[88]:
```
array([[0.0e+00, 1.0e+00, 0.0e+00, 3.0e+01, 5.4e+04],
       [1.0e+00, 0.0e+00, 0.0e+00, 3.7e+01, 6.7e+04]])
```

[89]:
```
y_test
```

[89]: array([0, 1])

0.0.7 Feature scaling and data normalization

[90]:
```
from sklearn.preprocessing import StandardScaler
```

[91]:
```
sc_X = StandardScaler()
```

[92]:
```
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

[93]:
```
X_train
```

[93]:
```
array([[-0.77459667, -0.57735027,  1.29099445, -0.19159184, -1.07812594],
       [-0.77459667,  1.73205081, -0.77459667, -0.01411729, -0.07013168],
       [ 1.29099445, -0.57735027, -0.77459667,  0.56670851,  0.63356243],
       [-0.77459667, -0.57735027,  1.29099445, -0.30453019, -0.30786617],
       [-0.77459667, -0.57735027,  1.29099445, -1.90180114, -1.42046362],
       [ 1.29099445, -0.57735027, -0.77459667,  1.14753431,  1.23265336],
       [-0.77459667,  1.73205081, -0.77459667,  1.43794721,  1.57499104],
       [ 1.29099445, -0.57735027, -0.77459667, -0.74014954, -0.56461943]])
```

[94]:
```
X_test
```

[94]:
```
array([[-0.77459667,  1.73205081, -0.77459667, -1.46618179, -0.9069571 ],
       [ 1.29099445, -0.57735027, -0.77459667, -0.44973664,  0.20564034]])
```

0.1 After completeing the above steps

0.1.1 it feels like most of this process can be automated as steps will remain same although there can be a need of parameter twiking

[ ]:
```
```